SAVE - Simulated Annealing Applied to the Vertex Elimination Problem in Computational Graphs

Uwe Naumann

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

SAVE - Simulated Annealing applied to the Vertex Elimination Problem in Computational Graphs

Uwe Naumann

N° 3660

April 1999

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

SAVE - Simulated Annealing applied to the Vertex Elimination Problem in Computational Graphs

Uwe Naumann *

Thème 2 — Génie logiciel
et calcul symbolique
Projet TROPICS

Rapport de recherche n° 3660 — April 1999 — 24 pages

Abstract: The chain rule - fundamental for Automatic Differentiation (AD) - can be applied to computational graphs representing vector functions in arbitrary orders resulting in different operations counts for the calculation of their Jacobian matrices. Very few authors have looked at this interesting subject so far and there is no generally accepted terminology for dealing with these combinations of the forward and reverse modes of AD. The minimization of the number of arithmetic operations required for the calculation of the complete Jacobian leads to a computationally hard combinatorial optimization problem.

In this paper we will describe a new heuristic approach to the solution of the vertex elimination problem in computational graphs which can easily adapted to the more general case of eliminating edges. Simulated annealing is widely regarded as a suitable method for solving combinatorial optimization problems. We will discuss different annealing schedules and present some test results. Finally we will regard this approach in comparison with other methods for computing Jacobians using a minimal number of arithmetic operations.

Key-words: Computational graph, vertex elimination, simulated annealing

* e-mail: Uwe.Naumann@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

SAVE - Recuit Simulé appliqué au Problème de l'Élimination des Noeuds du Graphe de Calcul

Résumé : Nous présentons une nouvelle approche de calcul de Jacobiennes basée sur l'élimination des arcs dans le graphe de calcul. L'objectif de cette méthode est la minimisation du nombre de multiplications scalaires nécessaires à l'accumulation de la Jacobienne d'une fonction vectorielle à un argument. Pour cela il faut résoudre un problème d'optimization combinatoire. La méthode de recuit simulé offre la possibilité de résoudre ce problème.

Mots-clés : Graphe de calcul, élimination des noeuds, recuit simulé.

Contents

1 Introduction

Research in the field of **Automatic Differentiation (AD)** dates back as far as 1964, when R. E. Wengert ([Wen64]) first proposed the automation of the calculation of derivatives by a program in form of the basic forward mode. Since then a lot of work has been done in order to make AD faster and more widely applicable (see [CoGr91] and [BBCG96]). AD is a fast and convenient way for calculating directional derivatives of vector functions numerically up to machine precision provided that it is given as a computer program. Notice, that AD is completely different from the approach to computing derivatives numerically through divided differences.

The computation of the Jacobian matrix of a vector function

$$F: \quad \mathbb{R}^n \supseteq D \to \mathbb{R}^m \quad : \quad \mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x}) \tag{1}$$

is essential in many numerical algorithms. For practical reasons most currently available AD software packages provide only two approaches to calculating the Jacobian matrix of F at the current argument – the forward and the reverse modes which are based on the application of the chain rule to F in two different ways. However, the chain rule can be applied to computational graphs of vector functions in any arbitrary order, which results in different operations counts for the calculation of the Jacobian matrix J. The general task of efficiently evaluating J using an approach which is sometimes referred to as *cross-country elimination* is conjectured to be NP-hard. The fact that Jacobians can be calculated by eliminating intermediate vertices in computational graphs is well known since the late 1980's. There are a few papers by various authors that motivate a closer look at this topic [GrRe91], [Bis96]. However, there is no generally accepted terminology for dealing with these combinations of the forward and reverse modes of AD, so far.

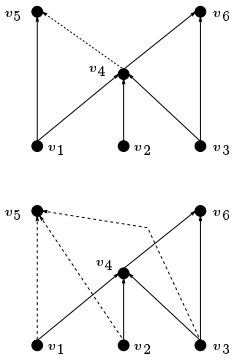

Figure 1: $(4,5)$

In our approach we expect the vector function F to be such that it can be decomposed into a sequence of scalar valued functions $\varphi : \mathbb{R}^d \supseteq D_\varphi \to \mathbb{R}$ that take a vector $\mathbf{u} \in \mathbb{R}^d$ as their argument and return a value $w = \varphi(\mathbf{u})$. We call such functions **elemental**. In most cases the vector function F is given as a computer program written in some high-level programming language such as `C`, `C++` or `Fortran`. This specification of F is called **evaluation routine** if it can be broken down into a sequence of scalar assignments of the form $(\mathbb{R} \ni)v_j = \varphi_j(v_i)_{i \in P_j}$ by assigning the result of every elemental function φ_j that occurs in the program to a unique intermediate variable v_j. Here P_j is the index set of the arguments of φ_j and we denote its cardinality by $|P_j|$. Since our objective is to calculate the complete Jacobian of a given vector function which consists of the partial derivatives of the m dependent variables $y_0, \ldots, y_{m-1}$ with respect to each of the n independent variables $x_0, \ldots, x_{n-1}$ it is fundamental to assume the following:

Assumption 1.1 *Given an evaluation routine of a vector function defined by Equation (1), for some fixed argument the elemental functions φ_j are well defined for $j = 1, 2, \ldots, q+m$ and have jointly continuous partial derivatives*

$$c_{ji} \equiv \frac{\partial}{\partial v_i} \varphi_j(v_k)_{k \in P_j} \quad \textit{for} \quad i \in P_j$$

of their respective arguments on some neighborhood $\mathcal{D}_j \subset I\!R^{n_j}$ with $n_j \equiv |P_j|$.

The relation between the variables in a given evaluation routine can be visualized by a directed acyclic graph $CG = (V, E)$ which contains all information needed to be able to compute the entries of the Jacobian. From now on we will refer to CG as the **computational graph** (also **c-graph**) We will distinguish between $n \equiv |X|$ minimal [independent], $p \equiv |Z|$ intermediate and $m \equiv |Y|$ maximal [dependent] vertices:

$$X \equiv \{v_{1-n}, \ldots, v_0\}, \quad Z \equiv \{v_1, \ldots, v_p\}, \quad Y \equiv \{v_{p+1}, \ldots, v_{p+m}\}.$$

There exists a directed edge (i, j) connecting a vertex v_i with a vertex v_j in CG if v_j directly depends on v_i in F. As above we set P_j $[S_j]$ to be the set of indices of all predecessors [successors] of a vertex v_j. According to Assumption 1.1 labels c_{ji}, representing the local partial derivatives, are attached to all edges $(i, j) \in E$.

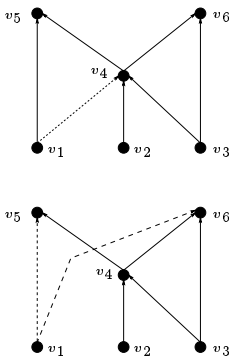

Figure 2: $(1, 4)$

We are looking for a method of transforming the c-graph of F such that we get the Jacobian J at the lowest possible cost in terms of the number of scalar multiplications involved in this process. In fact, if by successively eliminating all vertices representing intermediate variables in the underlying evaluation program (which is equivalent to eliminating all edges having either an intermediate vertex as source, or having such a vertex as target, or both, i.e. all *intermediate edges*) we get to a stage, where the c-graph represents a subgraph of the complete bipartite graph $K_{n,m}$ and the labels $c_{ji} \equiv \partial y_j / \partial x_i$ (i=0, ... ,n-1, j=0, ... ,m-1) on the edges connecting the minimal vertices with the maximal ones, are exactly the nonzero entries of the Jacobian matrix of F. The elimination of intermediate edges represents the elemental action that we will build on in our approach. It can be regarded as the the chain rule applied to evaluation routines in form of c-graphs. Consequently we distinguish between two ways to eliminate an edge in CG and we will refer to them as **forward** and **backward** edge elimination.

Graphically, the forward elimination of an edge (i, j) is equivalent to connecting all predecessors of v_i with v_j (provided they have not been connected before as we do not permit multiple edges) followed by updating the existing or generating the new local partial derivatives and, finally, the deletion of (i, j). This is illustrated in Figure 1 for the edge $(4, 5)$. In correspondence with the chain rule we multiply the values of successive edges (i, j) and (j, k) whereas we add the values of parallel edges having the same source and the same target. Thus, the forward elimination of an edge (i, j) involves a number of scalar

multiplications that is equal to the cardinality of the predecessor set of its source v_i. We will call this number the **in-degree** or **forward Markowitz degree** of (i,j) and denote it by $|P_{(i,j)}| = |P_i|$.

The graphic interpretation of the backward elimination of $(1,4)$ is shown in Figure 2. As in the case of forward elimination we simply have to insert new edges connecting v_l with all successors of v_i (if they do not exist already) and generate new or update the existing edge labels correspondingly. Finally, (l,i) is removed from the c-graph. It takes $|S_{(i,j)}|$ scalar multiplications to eliminate an edge (i,j) backward. $|S_{(i,j)}|$ denotes the **out-degree** (**backward Markowitz degree**) of (i,j) which is equal to the number $|S_j|$ of successors of the target v_j.

Let the length of a path connecting an independent vertex with a dependent one be defined as the number of edges it consists of. Then one can show that the sum of the total lengths of all distinct paths in the c-graph is strictly monotonically decreasing during the process of eliminating edges. We may conclude that edge elimination will always terminate. This seems to be obvious but it is certainly crucial for our approach.

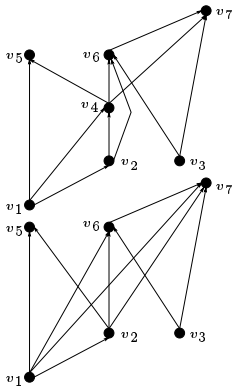

Figure 3: v_4

The forward [backward] elimination of all out-edges [in-edges] of a vertex v_i leads to the elimination of v_i itself (Figure 3). Consequently, the elimination of an intermediate vertex v_i involves $|P_i| \cdot |S_i|$ scalar multiplications which is usually referred to as the **Markowitz degree** of v_i [GrRe91]. So far, the application of the chain rule to c-graphs has been interpreted as vertex elimination. Even the few attempts to minimize the number of multiplications needed to calculate the Jacobian are based on the idea of eliminating vertices (see for example [Bis96] or [GrRe91]). However, there are problems where the optimal vertex elimination sequence does not minimize the number of multiplications. Let us illustrate this with the help of an example displayed in Figure 4. It shows a c-graph of a problem with two independent, five dependent, and two intermediate variables. There are two different vertex and numerous different edge elimination orders. The two vertex elimination orders result in $((v_i,v_k)\hat{=}5+10=)15$ and $((v_k,v_i)\hat{=}4+10=)14$ multiplications. So, using a pure vertex elimination strategy on this very simple example gives us the Jacobian for a cost of at least 14 multiplications. Now, suppose we eliminate (i,j) separately before v_k followed by the elimination of v_i. This would take only 13 multiplications which is obviously less than $14 = \min\{(v_i,v_k),(v_k,v_i)\}$ in the pure vertex elimination mode. In [Nau99] we have given an example for this. Furthermore, we have shown that the **vertex-edge discrepancy** does not exceed a factor of $(\sqrt{2}(2-\sqrt{2}))^{-1}$ for c-graphs containing two intermediate vertices. The problem remains unsolved for the general case involving c-graphs with $p > 2$ intermediate vertices. However, building on numerous test results we conjecture that the vertex-edge discrepancy will be less or equal to 2 for the general case.

For a Jacobian matrix J associated with a given vector function F we are going to think about a way to compute all its entries at a minimal cost. For reasons described in [Nau99] we have decided to regard the number of multiplications required to compute the complete

Jacobian as the cost and we will denote this objective function by $\mathbf{Cost}\{J\}$. Specifically, we will take the c-graph CG of F and we will search for an edge elimination sequence that minimizes $\mathbf{Cost}\{J\}$.

The successive elimination of edges from the c-graph defines the so-called **metagraph**

$$M = M(CG) = (V_M, E_M)$$

the vertices $w_k \in M$ of which represent all different c-graphs that could possibly be derived from CG by edge elimination. Labeling the edges in M with the cost of getting from the graph represented by their source to the one associated with their target we end up with a shortest path problem on the metagraph. The edge labels are considered to be the distances and we will refer to this problem as the **general edge elimination problem**. The difficulties arise from the fact that both the number of vertices and the number of edges in the metagraph depend on the number of intermediate vertices in the original c-graph exponentially. Therefore, an exhaustive search as well as any algorithm for computing a shortest path in M are not practicable. In order to decrease the complexity of the problem to solve we have to make certain restrictions, thus reducing both size of the metagraph defined by the number of its stages and the number of different paths to check. This approach will lead to subgraphs which are then subject to an analogous shortest path problem. With every restriction ρ we can associate a corresponding minimal cost $\mathbf{Cost}_\rho\{J\}$ of computing the Jacobian by solving the shortest path problem on the induced subgraph $M_\rho \subseteq M$ of the metagraph, i.e. $\mathbf{Cost}_\rho\{J\} = c_\rho \cdot \mathbf{Cost}\{J\}$ for some $c_\rho \geq 1$. Obviously, we would like the factor to be as small as possible as for large values c_ρ the chosen restriction ρ would not be very useful. One possibility is the restriction to eliminating vertices in the c-graph which will be exploited in this paper. This approach leads to the so-called **vertex metagraph** $M_\mathbf{V} \subseteq M$ an instance of which is shown in Figure 5. For further information on the general edge elimination problem refer to [Nau99].

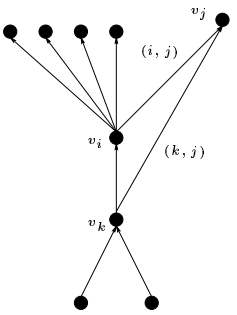

Figure 4: Surprise

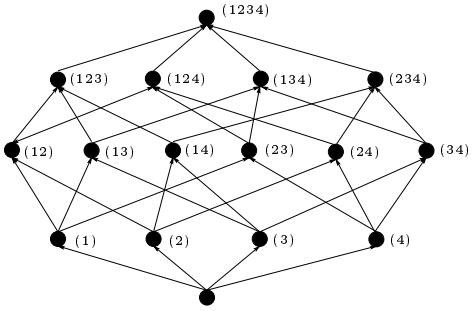

Figure 5: Vertex Metagraph $M_\mathbf{V}$

We have developed new and adapted some well-known algorithms for solving the combinatorial optimization problem. These methods contain local heuristics as well as dynamic programming algorithms and a collection of simulated annealing schedules for the computation of nearly optimal vertex elimination sequences. The latter will be looked at more closely in this paper. All the approaches proposed are implemented as parts of a C++ program named **OESCOMP**, which all our test results will be based on.

Considering local heuristics for edge or vertex elimination in c-graphs there are, obviously, two

trivial cases: Let $v_i < v_j$ denote the situation where a vertex v_i is eliminated before v_j. The elimination order yielding

$$\forall w_k \in M: \qquad v_i < v_j \quad \Leftrightarrow \quad i < j$$

will be referred to as the **forward vertex elimination mode (V_F)**. Analogous, we define the **backward vertex elimination mode (V_B)**. Both **V_F** and **V_B** are considered to be equivalent to the corresponding edge elimination modes. Since our objective is to minimize the cost of computing the complete Jacobian it certainly makes sense to think about how cheaply a particular intermediate vertex v_i can possibly be eliminated. The logical result would be to order all intermediate vertices increasingly by their Markovitz degrees at each stage of the metagraph. At a particular stage $w_k \in M_{\mathbf{V}}$ we eliminate the vertex with the lowest Markovitz degree among all intermediate vertices. This approach is known as the **Lowest-Markovitz-Degree-First (V_LM)** strategy:

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad |P_i|_k \cdot |S_i|_k \ \leq \ |P_j|_k \cdot |S_j|_k.$$

Here $|P_i|_k$ [$|S_i|_k$] denotes the number of predecessors [successors] of a vertex $v_i \in CG$ at a certain stage $w_k \in M_{\mathbf{V}}$ in the (vertex) metagraph. As a well-known heuristic for the minimization of the generated fill-in during the solution of sparse systems of linear equations the Markovitz based approach to the vertex elimination problem in c-graphs has already been examined in several papers like for instance in [GrRe91]. However, numerous tests showed that, in general, **V_LM** does not deliver optimal elimination sequences.

Let us consider an improved version of the **V_LM** heuristic. We define the **input-dependency degree** of a vertex $v_j \in Y \cup Z$ as $\mathtt{id}_j = k$ if there are paths connecting a maximum of k minimal vertices with v_j. Analogous the **output-dependency degree** of a vertex $v_j \in X \cup Z$ is $\mathtt{od}_j = k$ if a maximum of k maximal vertices are reachable from v_j. For a vertex $v_j \in Z$ we define its **dependency degree** as $\mathtt{dd}_j \equiv \mathtt{id}_j \cdot \mathtt{od}_j$. The dependency degree of a vertex is invariant with respect to the pure vertex elimination strategy which makes it ideal for the implementation of new heuristics. In particular, we have implemented the

Lowest-Relative-Markowitz-Degree-First heuristic (V_LR):

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad |P_i|_k \cdot |S_i|_k - \mathtt{dd}_i \ \leq \ |P_j|_k \cdot |S_j|_k - \mathtt{dd}_j.$$

Numerous test showed that this heuristic is superior to **V_LM** in most cases. We will illustrate this using an example given by the c-graph displayed in Figure 6. Obviously, it is easy to solve the resulting vertex elimination problem in three intermediate variables. It is even possible to check all different orders. Notice that **V_LM** does not deliver the minimum in this very simple case. It behaves like **V_F** resulting in 22 multiplications. Considering **V_LR** we have $\mathtt{dd}_1 = 6$, $\mathtt{dd}_2 = \mathtt{dd}_3 = 9$ and since $6 - 9 < 4 - 6$ the vertex v_2 is eliminated first. At the next stage we get $6 - 9 < 8 - 6$ and therefore **V_LR** would act the same way as **V_B** delivering the minimal operations count which is 18 for this example.

Considering strategies for edge elimination we have, among others, proposed the class of fill-in based heuristics. The fill-in generated by the elimination of an intermediate edge (i, j) is defined as the number of edges in the c-graph after minus the number of edges before the elimination. Every edge labeled with an elementary partial derivative is a potential factor in the elimination process which represents the computational process of accumulating the complete Jacobian. A logical consequence of these observations is to keep the number of these factors as low as possible which implies the minimization of the locally produced fill-in. This idea is exploited in the **Lowest-Fill-in-First** edge elimination heuristic (**E_LF**). Here, we always eliminate the edge producing the lowest fill-in next.

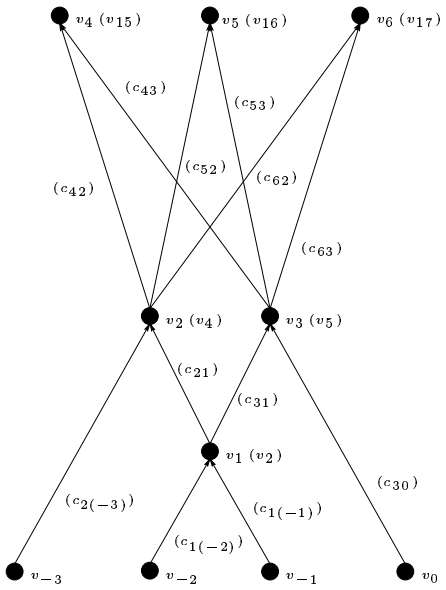

Figure 6: CG

A different approach to solving the general edge elimination problems is built on the representation of the Jacobian as chained matrix products. The algorithm which will be denoted by **DP** is described in [Nau99].

Edge elimination in c-graphs makes full use of the structural sparsity of the given problem. It is often possible to reduce the number of multiplications required to accumulate the complete Jacobian by a factor of three and more compared to the method proposed by Newsam and Ramsdell [NeRa83]. The savings compared to the dense forward and reverse modes are even more significant. When presenting test results in Section 5 we will consider the achieved values relative to lower bounds for values of the best choice out of dense forward and reverse modes (**DM**), i.e.

$$\mathbf{Cost_{DM}}\{J\} = \min\{n(m+p), m(n+p)\},$$

and the corresponding minimum operations count achieved by a uni-directional approach of the method by Newsam and Ramsdell (**NR**) which is

$$\mathbf{Cost_{NR}}\{J\} = \min\{\hat{n}(m+p), \hat{m}(n+p)\}.$$

2 Simulated Annealing

Simulated annealing is a global optimization method that distinguishes between different local optima. Starting from an initial point, the algorithm takes a step and the function is evaluated. When minimizing a function, any downhill step is accepted and the process repeats from this new point. An uphill step may be accepted. Thus, the algorithm can escape from local optima. This uphill decision is made by the Metropolis criterion. As the optimization process proceeds, the length of the steps decline and the algorithm closes in on the global optimum. Since the algorithm makes very few assumptions regarding the function to be optimized, it is quite robust with respect to combinatorial problems. The degree of robustness can be adjusted by the user.

In general, combinatorial optimization problems are characterized by an objective function to be minimized (the overall Markovitz degree in our case) and a discrete, very large configuration space (the $p!$ different orders in which we can eliminate the p intermediate vertices in the c-graph). Furthermore, the concept of *direction* telling us about *going downhill* or not does not exist in our context.

The roots of the simulated annealing algorithm lie in metallurgy. During the annealing process of metal its atoms form crystals. An optimal crystal is one with a configuration of lowest energy E. If we run the cooling process too fast then the crystals will not reach the optimal configuration. On the other hand, we do not want the annealing process to take too long. The goal is to find a useful compromise between the robustness of the algorithm and its speed. This is done by varying certain parameters within the algorithm.

Let us assume that we are looking for the configuration which minimizes a certain cost function. The algorithm can then be formulated as follows: Starting off at an initial configuration, a sequence of iterations is generated. Each iteration consists of the random selection of a configuration from the neighborhood of the current configuration and the calculation of the corresponding change in the cost function. The neighborhood is defined by the choice of a generation mechanism, i.e. a "prescription" to generate a transition from one configuration into another by a small perturbation. If the change in the cost function is negative the transition is unconditionally accepted. If the cost function increases the transition is accepted with a probability based upon the Boltzmann distribution

$$P(E) \quad \sim \quad e^{\frac{E}{k \cdot T}}$$

where k is a constant and the temperature T is a control parameter. This temperature is gradually lowered throughout the algorithm from a sufficiently high starting value (i.e. a temperature where almost every proposed transition, both positive and negative, is accepted) to a "freezing" temperature where no further changes occur. In practice, the temperature is decreased in stages and at each stage the temperature is kept constant until thermal quasi-equilibrium is reached. The whole of parameters determining the temperature decrement (initial temperature, stop criterion, temperature decrement between successive stages, number of transitions for each temperature value) is called the cooling schedule.

Consequently the four key ingredients for the implementation of simulated annealing are:

1. the definition of configurations of the system;
2. a generation mechanism, i.e. the definition of a neighborhood on the configuration space; a generator of random changes in the configuration;
3. the choice of a cost function (analog of energy) the minimization of which is the goal of the procedure;
4. a control parameter T and a cooling schedule telling us after how many random changes in configuration each downward step in T is taken. The assignment of this schedule requires deeper insight into the problem and trial-and-error experience.

The traveling salesman problem

In [PTVF92] we have found a concrete illustration of how to approach combinatorial optimization problems by simulated annealing. The *traveling salesman problem* (TSP) belongs to the class of NP-complete problems whose computation time for an exact solution grows exponentially with its size. For an exhaustive discussion of NP-completeness refer to [GaJo79].

The TSP may be regarded as the "mother" of all transport optimization problems. A salesperson visits n cities with given coordinates (x_i, y_i), returning finally to his city of origin. Each city is to be visited only once while making the route as short as possible. This leads to the objective function

$$E = \sum_{i=1}^{n} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \qquad ((x_{n+1}, y_{n+1}) \equiv (x_1, y_1)) \tag{2}$$

which is taken just as the total length of the journey. The above is equivalent to finding a *Hamilton cycle* of minimal length in the complete graph K_n with edges representing the distances between the cities. The vertex elimination problem in a c-graph is very similar to the *TSP*. Thus, many algorithms that work well for the latter can often easily be adapted to serve our purposes.

3 Application to the Vertex Elimination Problem

As a problem in simulated annealing the vertex elimination problem is handled as follows:

3.1 Configuration

The vertices are numbered $i = 1, \ldots, p$ and each of them has a Markovitz degree $|P_i|_k \cdot |S_i|_k$ depending on the stage w_k in the metagraph. A configuration is a permutation of the indices $1, \ldots, p$ interpreted as the order in which the intermediate vertices are eliminated. The key

difference between our problem and the traveling salesman problem is that the Markovitz degree is not a static value. It rather depends on the vertices that have been eliminated so far. However, in the TSP every city has its static coordinates making this problem even "easier" to solve. (This is not to be taken literally as the TSP is proven to be NP-hard ...)

3.2 Rearrangements

We use a slightly altered version of the moves suggested by Lin ([Lin65]). There are two types of moves:

1. If $(i_1, \ldots, i_p)$ is the current elimination sequence then we remove a dense subsequence $(i_j, \ldots, i_{j+k})$ and replace it with itself in the opposite order making

   $$(\ldots, i_{j-1}, i_{j+k}, \ldots, i_j, i_{j+k+1}, \ldots)$$

   the next elimination sequence to be regarded or

2. a dense subsequence $(i_j, \ldots, i_{j+k})$ is removed and then replaced in between two indices i_l, i_r on another, randomly chosen, part of the elimination sequence, i.e. $(i_1, \ldots, i_p)$ becomes

   $$(\ldots, i_{j-1}, i_{j+k+1}, \ldots, i_l, i_j, \ldots, i_{j+k}, i_r, \ldots)$$

   provided that $l > j + k$. Otherwise, we get

   $$(\ldots, i_l, i_j, \ldots, i_{j+k}, i_r, \ldots, i_{j-1}, i_{j+k+1}, \ldots)$$

   for $r < j$. We do not permit other rearrangements apart from these two.

3.3 Objective function

We intent to minimize the cost of accumulating the complete Jacobian matrix of a vector function by solving the shortest path problem in the metagraph $M_{\mathbf{V}}$ which is induced by the restriction to pure vertex elimination. Since the Markovitz degree of a vertex is dynamically changing throughout the elimination process its calculation is rather expensive ($O(p)$) having in mind that we will have to compute it numerous times during the simulated annealing algorithm. Notice that the computation of the Euclidean distance between two cities in the TSP which is part of Equation (2), is comparably cheap.

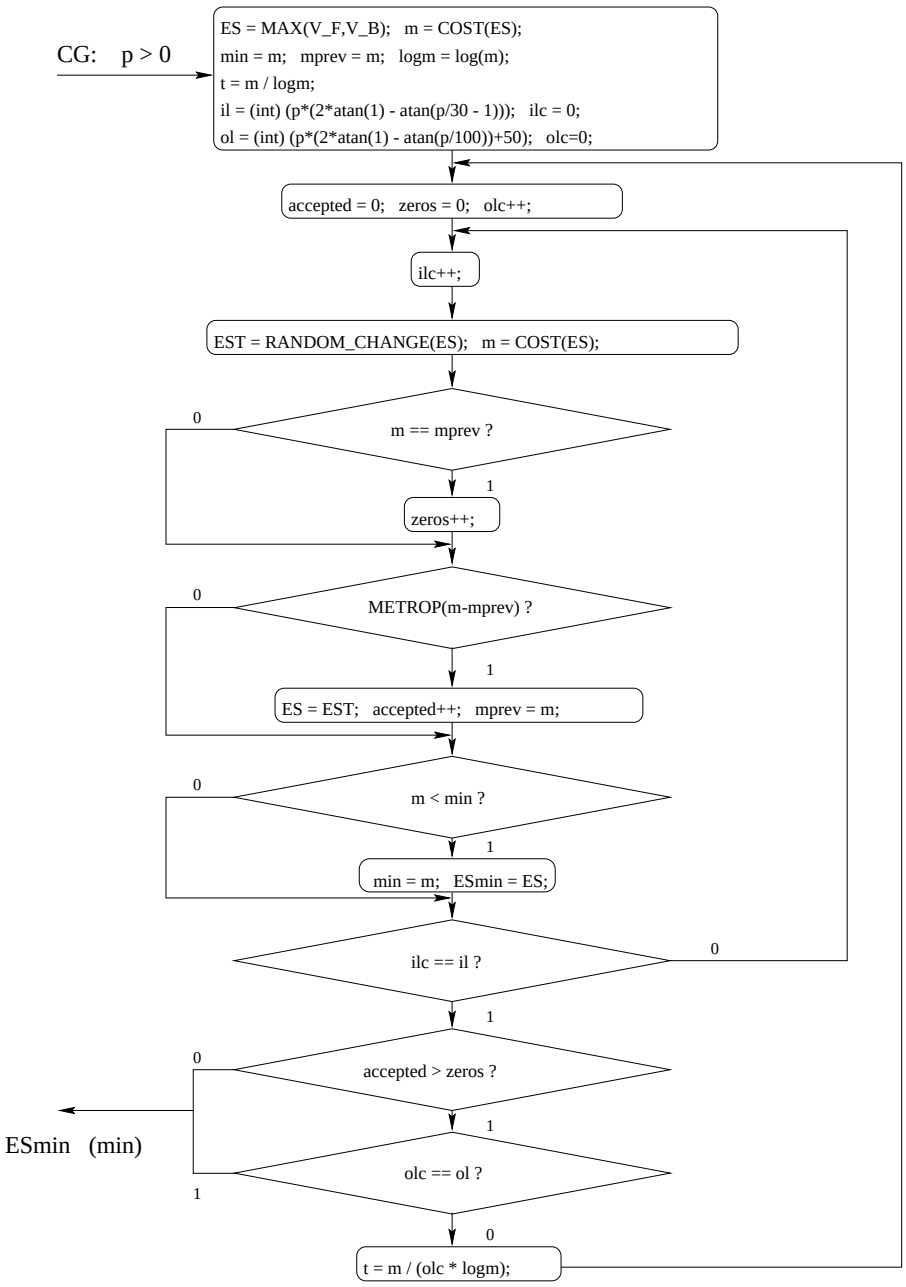

Figure 7: Simulated Annealing Algorithm

3.4 Annealing schedule

Figure 7 shows how the implemented simulated annealing algorithm works. It starts with an initial elimination sequence (`ES`) which we have chosen to be the maximum out of forward and backward vertex elimination modes in terms of the number of multiplications required for the accumulation of the complete Jacobian. At the beginning of the optimization procedure we do not want to be the cost `m` of `ES` close to the minimum. In this case it would become very likely that the algorithm stops after just one iteration without delivering any improvement.

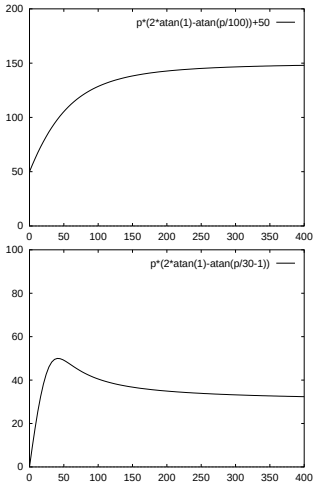

Figure 8: Loops

Our simulated annealing algorithm consists of two main loops – an outer loop (counter: `olc`) and an inner loop (counter: `ilc`). For both of them we define upper bounds for the number of iterations they will perform (`ol` and `il`). `il*ol` is the maximal number of iterations (the maximal number of elimination sequences that are generated and run) before the algorithm stops, even if it has not converged. It may happen that one and the same elimination sequence is checked twice or even more often, although it is very unlikely. We make sure that we get a result within a reasonable time span by setting `ol` and `il`. Its quality strongly depends on the parameters of the simulated annealing algorithm.

Figure 8 shows the development of `il` and `ol` depending on the number `p` of intermediate variables in the c-graph CG. Notice that for our algorithm we always assume that there is at least one vertex to be eliminated from CG. The maximal number of cooling steps lies always between 50 and 150. It increases rapidly for small values of `p`, whereas it converges to 150 for larger graphs. Analogous, we observe a steep ascent in the first part (`0<p<50`) of the curve for `il`. Again, it settles for a value around 40 for increasing numbers of intermediate variables. Choosing both values as we did assures that the algorithm will always terminate after a reasonable and in most cases sufficient run-time.

The "temperature" `t` is lowered with every iteration of the outer loop and it is kept constant while running through the inner loop. Within the latter we randomly change the elimination sequence using the two *rearrangements* suggested above. Whether this new, so far temporary, order (`EST`) is accepted or not depends on the Metropolis criterion. It returns a boolean variable which issues a verdict on whether to accept a reconfiguration leading to a change `d=m-mprev` in the objective function `COST`. If `d<=0` `EST` will always be accepted. If `d>0` the answer of `METROP(d)` will only be positive with probability `exp(-d/t)`. As we have already pointed out, `t` is not changed within the inner loop. However, with every iteration (`olc`) of the outer loop we reinitialize `t` as `t=m/(olc*logm)`. Notice that the new value of `t` depends on the current Markovitz degree, i.e. it depends on the cost of the last accepted

elimination sequence. With this approach we get

$$t_j = \frac{m_{j-1}}{(j-1)\cdot\log(m_0)} \quad \Rightarrow \quad t_{j+1} = \frac{m_j}{j\cdot\log(m_0)}.$$

From the above we can derive the **cooling rate** Δ_t:

$$\Delta_t = \frac{t_{j+1}}{t_j} = \frac{(j-1)\cdot m_j}{j\cdot m_{j-1}}.$$

Why have we chosen to let `t` develop this way? Suppose we have taken a step upwards (accepted an elimination sequence which results in an increase of the objective function), i.e. $m_j > m_{j-1}$. Then we do not want to continue the cooling process at the same speed as before since this could lead to being unable to leave the current "valley". So, the temperature is permanently lowered as a result of the increasing outer loop counter `olc`. However, the extend up to which the system is cooled depends both on the current temperature and on the change of the Markovitz degree during the last iteration. This approach turned out to work well in most cases.

Throughout the entire annealing process we always keep track of the minimal Markovitz degree (`min`) resulting from any of the elimination orders that were checked so far. Thus, we do not entirely depend on the convergence of the algorithm. Even a very good elimination sequence which was found "by luck" in the high temperature phase of the annealing process can be the result of running the algorithm. After each outer loop iteration we check the *exit criterions*. There are three of them:

1. The maximal number of iterations to be performed is reached (`olc==ol`).
2. During the last outer loop iteration none of the rearrangements has been accepted.
3. We have chosen to accept all rearrangements which do not lead to any change in the cost function. However, if the only accepted rearrangements are those which are invariant with respect to the cost (`accepted == zeros`) we will also stop the algorithm.

Finally, the simulated annealing algorithm delivers the elimination sequence with the minimal cost, i.e. a vertex elimination order which approximates the minimal number of multiplications required for the accumulation of the complete Jacobian.

Summarizing the above we have the following list of parameters which can be experimented with when looking for an optimal annealing schedule:

- initial elimination sequence;
- initial temperature;
- cooling rate;
- types of rearrangements;

- number of iterations with constant temperature;
- number of cooling steps;
- acceptance philosophy.

There is certainly plenty of room for experiments. We have implemented four additional versions of our simulated annealing algorithm. In **SA_R** we use the reversal of dense subsequences as the only rearrangement action. **SA_T** is similar except for only transportation of dense subsequences being allowed. Furthermore, we have experimented with two different annealing schedules. **SA_CS** is similar to the described method with

$$t_j = \frac{m_{j-1}}{2j} \quad \Rightarrow \quad \Delta_t = \frac{t_{j+1}}{t_j} = \frac{j \cdot m_j}{(j+1) \cdot m_{j-1}},$$

i.e. we have slowed the cooling process down. In **SA_CF** we increase the temperature in the second step, which can be regarded as the generation of a new random initial elimination sequence, as we accept almost every rearrangement. Then we cool the system down with

$$t_j = \frac{m_{j-1}}{j^2} \quad \Rightarrow \quad \Delta_t = \frac{t_{j+1}}{t_j} = \frac{j^2 \cdot m_j}{(j+1)^2 \cdot m_{j-1}}$$

which is much faster then in the main method. In the algorithm described by Figure 7 we have used the following symbolism:

`ES, EST, ESmin` $\in \mathbb{N}^p$:	elimination sequences;
`m, mprev, min` $\in \mathbb{N} \cup \{0\}$:	cost values;
`il, ilc, ol, olc` $\in \mathbb{N} \cup \{0\}$:	for loops;
`t` $\in \mathbb{R}$:	temperature;
`logm` $\in \mathbb{R}$:	logarithm of initial cost;
`accepted, zeros` $\in \mathbb{N} \cup \{0\}$:	counters.

Now, it should be straight forward to follow the logic behind our simulated annealing algorithm.

4 Case Study

The success of simulated annealing depends on extensive tests and variation of parameters. Of course, we would like the algorithm to deliver nearly optimal results in virtually all cases without always having to adapt the parameters to the given problem. Furthermore, it should not run too long even for large-scale problems. We have applied **SAVE** to many test problems out of which we will discuss a small subset in detail. As supported by the results achieved our version of the simulated annealing algorithm performs well on a large number of problems. However, there are certain exceptions indicating that it is impossible to come up with a strategy (annealing schedule) which is universally optimal.

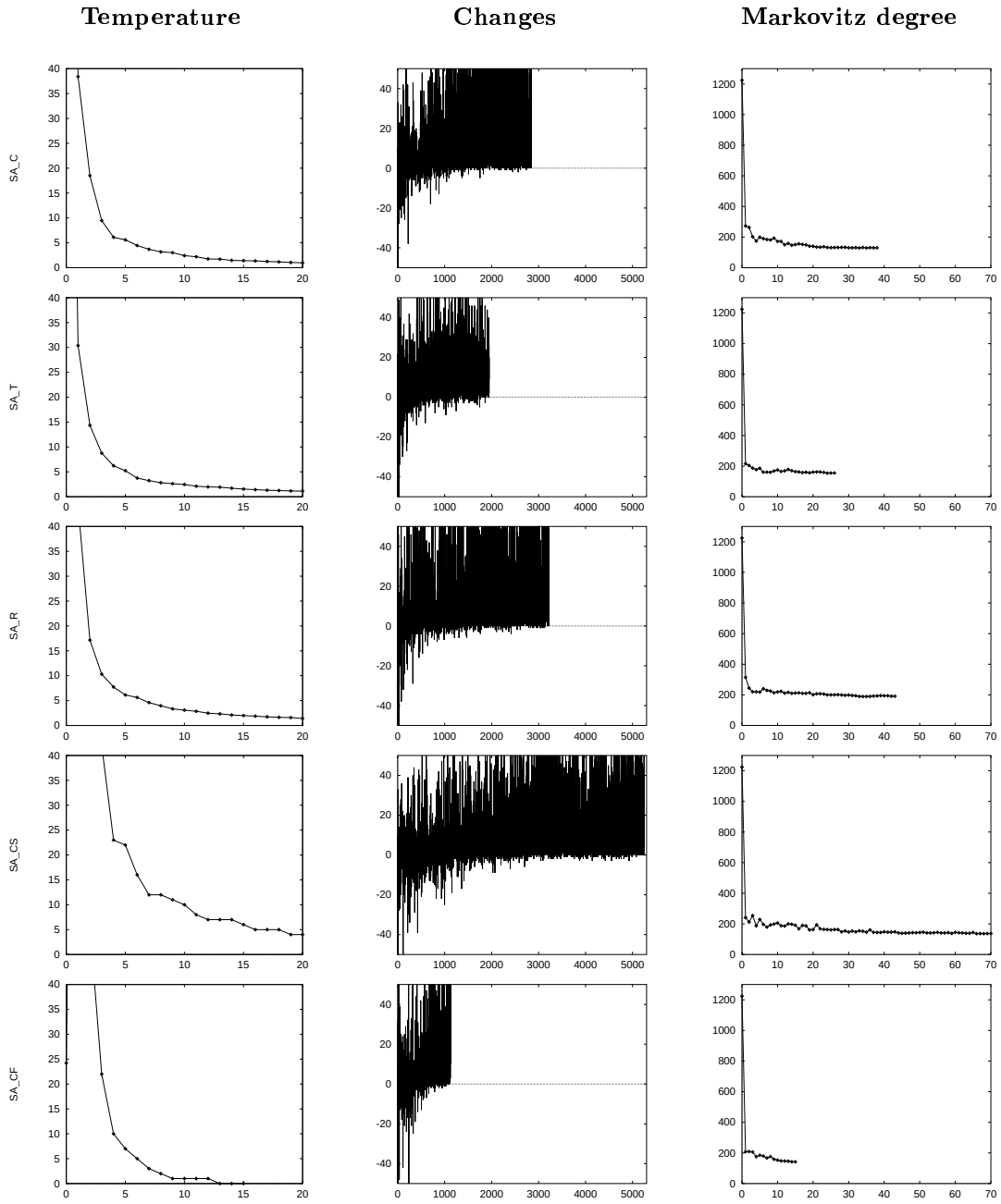

Figure 9: **SAVE** on SPF

4.1 Speelpenning function

The first test problem is due to Speelpenning [Spe80] and represents a simple chained product of the n independent variables:

$$y = f(\mathbf{x}) = \prod_{i=0}^{n-1} x_i. \qquad \text{(SPF)}$$

For $n = 50$ the c-graph contains 48 intermediate vertices. It can be checked that **V_B** is optimal on this problem using 96 multiplications for the computation of the gradient of f. On the other hand, running **V_F** results in an overall Markovitz degree of 1224 as a consequence of the successive increase of the in-degrees of the intermediate vertices.

Using Figure 9 we will compare all variations of our simulated annealing algorithm which form the five rows of the table. The three columns show the development of the temperature over the first 20 steps, the changes in the objective function after each iteration, and the course of the overall Markovitz degree which we intent to minimize, respectively. We have plotted the values for the temperature and the Markovitz degree with respect to the outer loop iterations, whereas the changes in the cost are shown for every single elimination sequence that has been checked, i.e. for each iteration of the inner loop. Apart from differences in the run-time the five methods converged to different final values of the overall Markovitz degree:

	SA_C	**SA_T**	**SA_R**	**SA_CS**	**SA_CF**
`min`	129	154	189	136	142
$t_{\texttt{user}}$	93 sec	63 sec	105 sec	171 sec	37 sec

Furthermore, we observe the following:

1. Neither a higher nor a lower cooling rate lead to improvements in the value of the objective function. However, the run-time of **SA_CF** undercuts the one of **SA_C** by a factor of nearly 3. Still it delivers an acceptable result.

2. Starting with the **V_F**-based elimination sequence, the reversal of substrings could be expected to lead to a good solution in a short time. Obviously, this is not the case. Why? The reason lies in the structure of the Speelpenning function. Whenever we eliminate the vertices of a part of the c-graph backward this can be regarded as "good" for the value of our objective function. However, running forward is certainly bad. Now, if we allow the reversal of sub-strings of a given elimination sequences to be the only rearrangement in the annealing process this can lead to the repeated negation of savings made in the current step by the rearrangements to come.

3. Depending on the accepted rearrangements and the resulting Markovitz degree the temperature is lowered at varying speeds. Especially, this becomes clear when looking at the graphs for **SA_CS** in the fourth row of Figure 9.

4. The largest decreases in the objective function value are achieved in the high temperature phase. This is certainly not very surprising.

5. It might often be advantageous to choose a high cooling speed in order to get useful estimates for the savings that can be expected.

It is difficult to state something of general validity when speaking about the method of simulated annealing applied to computationally hard combinatorial optimization problems. However, the approach that we have chosen as our main method (**SA_C**) turned out to deliver the best results in virtually all cases.

4.2 Steady state combustion problem

The steady state combustion problem (SSC) is the variational formulation of the underlying boundary value problem [ACM91]. For the examination of the behavior of simulated annealing we have chosen the case with $n = 4$ independent variables. This leads to a relatively small c-graph containing 103 intermediate vertices which is suitable for having a closer look at our algorithm.

SA_C converged to an elimination sequence that took 123 multiplications to compute the gradient. The **V_B**-based sequence (142 multiplications) served as the starting point. The best known value of the objective function achieved by **V_LR** as well as the dynamic programming approach and the method **SA_CS** is 122. Figure 10 shows the courses taken by the temperature t and by the Markovitz degree m_j.

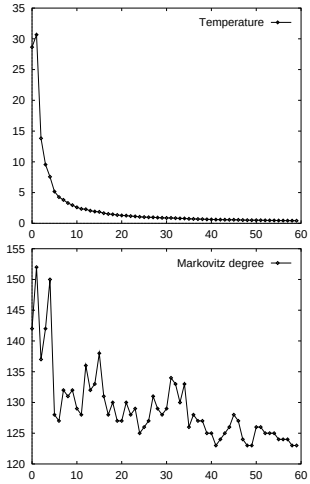

Figure 10: SSC

With `il=47` and `ol=130` the algorithm had to terminate after checking 6110 vertex elimination sequences. In fact, it performed only 59 outer loop iterations corresponding to the generation of 2773 (not necessarily different) elimination orders after which one of the *exit criterions* described in Section 3 was met. Exploiting one of the advantages of our simulated annealing algorithm the Markovitz degree is increased repeatedly, thus, being able to escape from local minima.

Obviously, the savings which can possibly be achieved are not so remarkable that they could justify the effort. It makes sense to apply simulated annealing to the vertex elimination problem in c-graphs if we care about the result only and ignore the time that it took to compute the elimination sequences. In most cases heuristics will deliver sequences which are nearly as good (or better) at a much higher speed.

4.3 Chebyshev quadrature problem

Both the SSC and the Chebyshev quadrature problem (CQ) are taken from the `MINPACK` test problem collection [ACM91]. For the latter we have chosen the case $n = 15$ and $m = 16$ resulting in a c-graph of reasonable size. Remember that we always take either the **V_F**- or the **V_B**-based elimination sequence as an initialization of the simulated annealing algorithm, depending on which of them delivers the higher cost. Starting with a relatively "bad" initial elimination sequence turned out to be advantageous for the behavior of the algorithm in many cases. Now, this function represents a counterexample as the starting sequence is obviously "too bad" for a problem of the given size and structure. The operations count resulting from the application of the backward vertex elimination mode (8400) is about four times as large as the one of the forward mode (1980). The algorithm "cools the system down" to 4842 which is approximately half the number of multiplications compared to its starting value. However, it does not reached the value which was calculated for **V_F**.

Figure 11 shows the development of the Markovitz degree. After some ups and downs in the first section of the annealing process, caused by the initially high temperature, the cost decreases continuously, unfortunately, not reaching the minimum. As a way out we could think of slowing the cooling process down in order to allow more iterations to be performed. Representing this approach **SA_CS** results in a cost of 4343. In fact, we observe an improvement compared to **SA_C**, although it is not very encouraging when taking into account that it took about four times as long to achieve the improvement.

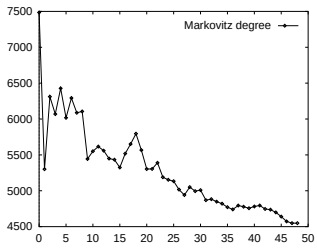

Figure 11: CQ

5 Tests, Conclusion and Outlook

We have applied the simulated annealing method to more than a hundred test problems out of which we will present a small but representative subset. The numbers of scalar multiplications resulting from different approaches will be listed for the following test problems:

`FDC`: Flow in a driven cavity problem;
`FCH`: Flow in a channel problem;
`WAT`: Watson function;
`DIE`: Discrete integral equation function;
`VDI`: Variably dimensioned function;
`GDF`: Gaussian data fitting problem.

The above examples are taken from the MINPACK test problem suite [ACM91]. In the following table we will compare the values delivered by the **SA_C** method with the theoretical operations counts that can be achieved using state-of-the-art AD-technology (see Section 1). Here, $\hat{n}$ [$\hat{m}$] denotes the maximal number of nonzero elements per row [column] in the Jacobian.

	n	p	m	$[\hat{n}, \hat{m}]$	**DM**	**NR**	**SA_C**	**NR / SA_C**
FDC	16	984	16	11	16000	11000	1234	**8.9**
FCH	32	1209	32	9	39712	11169	851	**13.1**
WAT	7	1683	7	7	11830	11830	5183	**2.3**
DIE	20	2499	20	20	50380	50380	1660	**30.3**
VDI	100	504	100	100	60400	60400	10337	**5.8**
GDF	11	1625	65	11	18590	18590	1477	**12.6**

Considering the ratio between the optimal one-sided Newsam-Ramsdell approach and the simulated snnealing method applied to vertex elimination we observe that, in fact, large savings can be achieved. Obviously, this cannot be the case for the computation of single gradients. However, for $\{n, m\} \gg 1$ we get a significant decrease of the number of multiplications involved in the accumulation of the Jacobian for virtually all problems.

The next table shows a comparison between the different approaches to the solution of the shortest path problem. We have also listed the values achieved by the better choice out of the forward and the backward vertex elimination modes, denoted as [**V_F**,**V_B**]. Notice that these results imply that we know which of the two methods is actually "the better choice". The differences between the results delivered by **V_F** and **V_B** can be very large in some cases.

	[V_F,V_B]	**V_LM**	**V_LR**	**E_LF**	**DP**	**SA_C**
FDC	930	1338	930	1106	930	1234
FCH	941	845	845	941	845	851
WAT	6473	4240	4240	4240	4240	5183
DIE	2059	1659	1659	2001	1659	1660
VDI	20400	10401	10301	103001	10301	10337
GDF	1430	1430	1430	1625	1430	1477

For most real-world problems the generation of optimized derivative code based on either forward or backward vertex elimination sequences combined with the pre-elimination of all hoisting vertices would result in remarkable savings in the overall operations count. The fact that it is not clear a priori whether we should prefer the forward or the backward approach is one of the problems. At this point it could be useful to exploit the strength of heuristics

for determining nearly optimal vertex elimination sequences for almost all sorts of c-graphs. **V_LR** turned out to be very consistent, while being only slightly more expensive than the pure uni-directional methods. It could be worth to analyze selected evaluation routines deeper by using more costly optimization methods like dynamic programming or simulated annealing. However, the additional effort is justified only if the resulting derivative code is generated once and is then used over and over again as the runtime of a simulated annealing algorithm is in general by magnitudes larger than a pure forward or backward approach.

To summarize the above, it appears to be useful to work on the automatic generation of optimized adjoint code based on different elimination sequences. In order to be able to handle large-scale evaluation programs the hierarchical approach has to be implemented efficiently. Therefore, we require tools for analyzing the code which generate some (ideally standardized) intermediate form which contains all the necessary information ([Bro98], [BRM96]).

The proof of the NP-completeness of the general edge elimination problem has not been given yet. Our conjecture about the small constant vertex-edge discrepancy has a more practical relevance. To support the search for its proof the implementation of a simulated annealing algorithm for optimizing edge elimination sequences could be useful. The principle is the same as for vertex elimination sequences. We simply have to adapt the annealing schedule such that rearrangements including both forward and backward elimination of edges become possible.

References

[ACM91] B. Averik, R. Carter, and J. More, *The Minpack-2 test problem collection (preliminary version)*, Technical Memorandum No. 150, Mathematical and Computer Science Division, Argonne National Laboratory, 1991.

[BBCG96] M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds, *Computational differentiation: techniques, applications, and tools*, SIAM, Philadelphia, PA, 1996.

[Bis96] C. H. Bischof, *Hierarchical approaches to automatic differentiation*, in [BBCG96], pp. 83–94.

[BRM96] C. Bischof, L. Roh, and A. Mauer, *ADIC: An extensible automatic differentiation tool for ANSI-C*, Preprint ANL/MCS-P626-1196, Argonne National Laboratory, March 1997.

[Bro98] S. Brown, *Models for automatic differentiation: A conceptual framework for exploiting program transformation*, PhD thesis, Computer Science, University of Hertfordshire, Hatfield, England, February 1998.

[CoGr91] G. Corliss and A. Griewank, eds, *Automatic differentiation: theory, implementation, and application*, SIAM, Philadelphia, PA, 1991.

[GaJo79] M. R. Garey and D. S. Johnson, *Computers and intractability - a guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[GrRe91] A. Griewank and S. Reese, *On the calculation of Jacobian matrices by the Markowitz rule*, in [CoGr91], pp. 126-135.

[Lin65] S. Lin, Bell System Technical Journal, Vol. 44, pp. 2245-2269.

[Nau99] U. Naumann, *Efficient calculation of Jacobian matrices by optimized application of the chain rule to computational graphs* Ph.D. thesis, Institute for Scientific Computing, Dresden University of Technology, 1999.

[NeRa83] G. Newsam and J. Ramsdell, *Estimation of sparse Jacobian matrices*, SIAM J. Alg. Discr. Meth., 4 (1983), pp. 404-417.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C*, Cambridge University Press, 1992.

[Spe80] B. Speelpenning, *Compiling fast partial derivatives of functions given by algorithms*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, January 1980.

[Wen64] R. E. Wengert, *A simple automatic derivative evaluation program*, Comm. ACM, 7 (1964), pp. 463-464.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399